Automatic Parallelization of NLPs with Non-Affine Index-Expressions

Marco Bekooij (NXP research)
Tjerk Bijlsma (University of Twente)

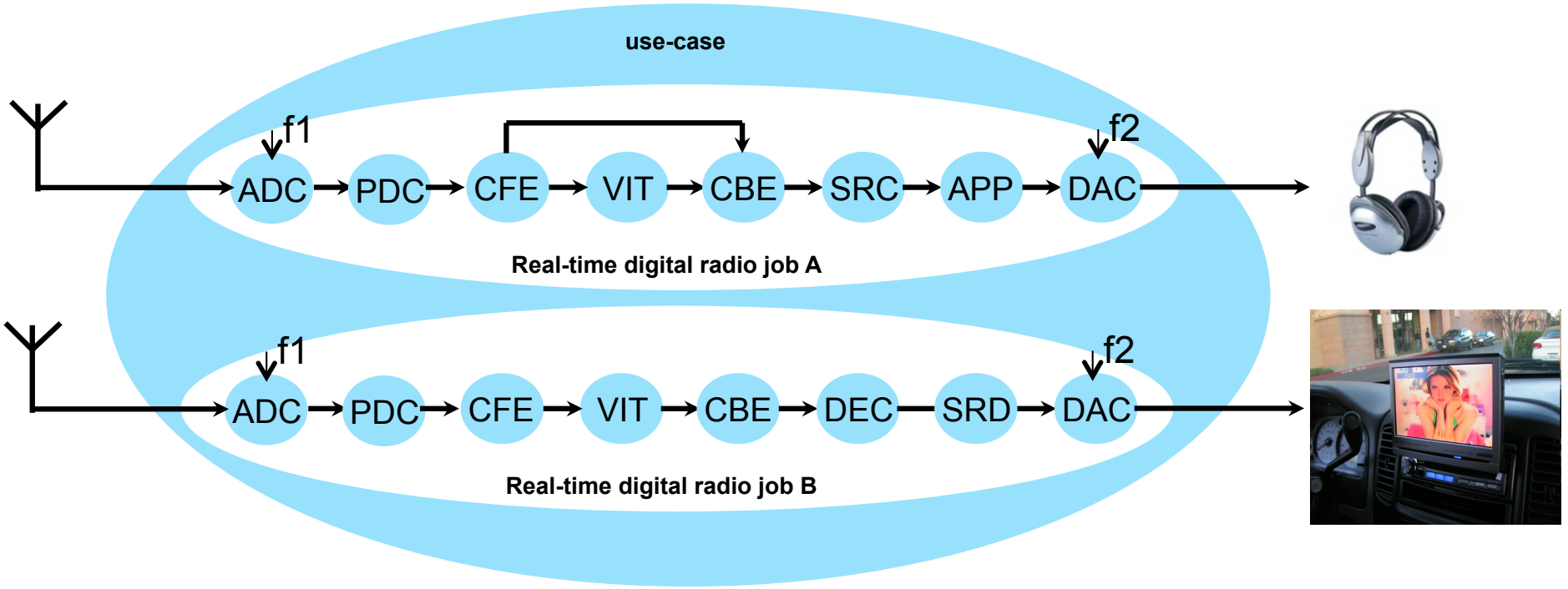
University of Twente
Enschede - The Netherlands

Outline

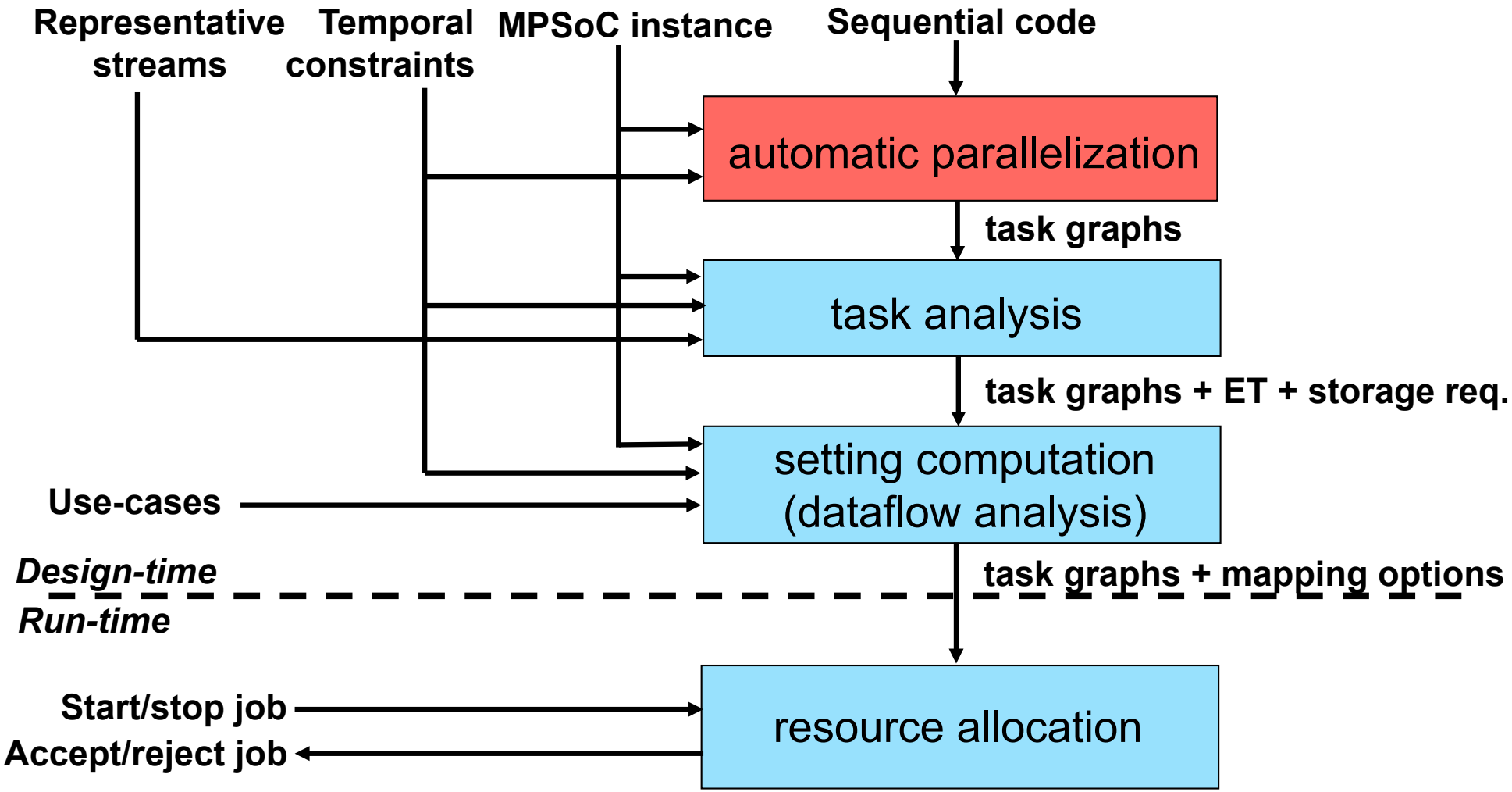
- Context: car-entertainment applications
- Mapping Flow
- Motivation
- Basic Idea
- Automatic Parallelization
  I. Extraction of tasks
  II. Addition of inter-task communication via CBs
  III. Computation of the window sizes
  IV. Insertion of communication and synchronization statements
  V. Computation of sufficient buffer capacities
- Conclusion

Multi-stream car-entertainment system

Car entertainment use-case



Mapping flow



Main reasons to extract task-level parallelism

- Meet the throughput constraint of the application
- Satisfy storage constraints embedded memories
- Correct by construction analysis model extraction

Important short-comings of existing parallelization techniques

- Restrictions on input language do not match well with multimedia stream processing domain:
  - Affine index-expression, for-loops containing side-effect free functions
  - Typically not well supported: if-conditions, while-loops, pointers, dynamic memory allocation, communication deep in function call hierarchy
- Parallelization is not driven by temporal & storage constraints
  - Missing underlying analysis model
- Observations:
  - Task-graph radio applications can often be represented as a dataflow model (e.g. VPDF and in some cases CSDF)
  - Dataflow model can be used to compute throughput, buffer capacities, scheduler settings, and synchronization granularity

Basic idea

- Input is single assignment nested loop program (NLP) with side-effect free functions
- Each function becomes a task
- Tool must insert synchronization calls and communication calls and must compute buffer capacities
- Prevent complex control by making use of circular buffers instead of FIFO buffers
  - Potentially multiple writing and reading tasks per buffer
  - Hide irregular access patterns in buffers with sliding windows

Automatic Parallelization

- Parallelization steps:
  I. Extraction of tasks

  II. Addition of inter-task communication via buffers

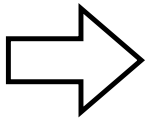
  III. Computation of the window sizes

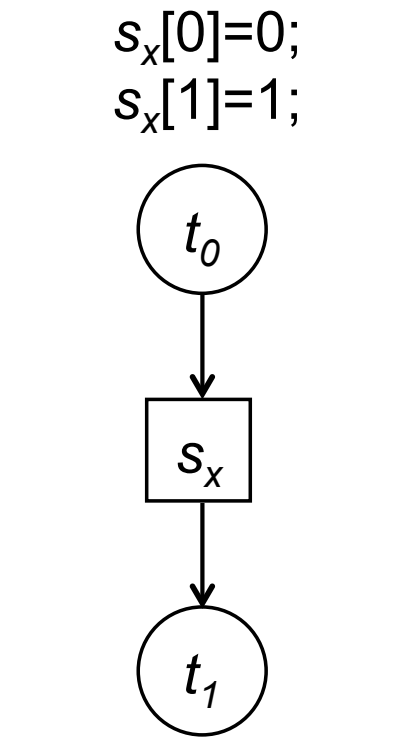
  IV. Insertion of communication and synchronization statements

  V. Computation of sufficient buffer capacities
     - A cyclo static dataflow model is used
       - To guarantee deadlock free execution
       - To compute buffer capacities

I. Extraction of tasks

```
X[0]=0;
X[1]=1;
for(i=0;i≤1;i++){
  F(x[i]);
}
```



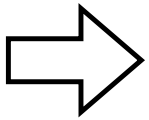


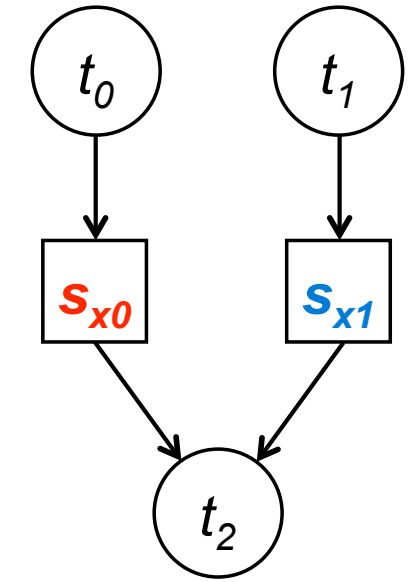

```
for(i=0;i≤1;i++){
  F(s_x[i]);
}
```

I. Extraction of tasks

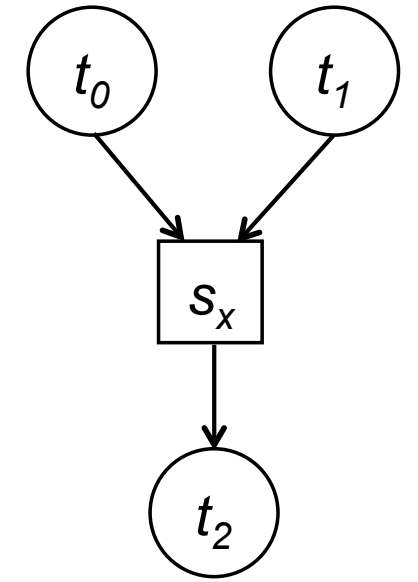
- We create a task out of each assignment-statement

```
X[0]=0;
X[1]=1;
for(i=0;i≤1;i++){
  F(x[i]);
}
```





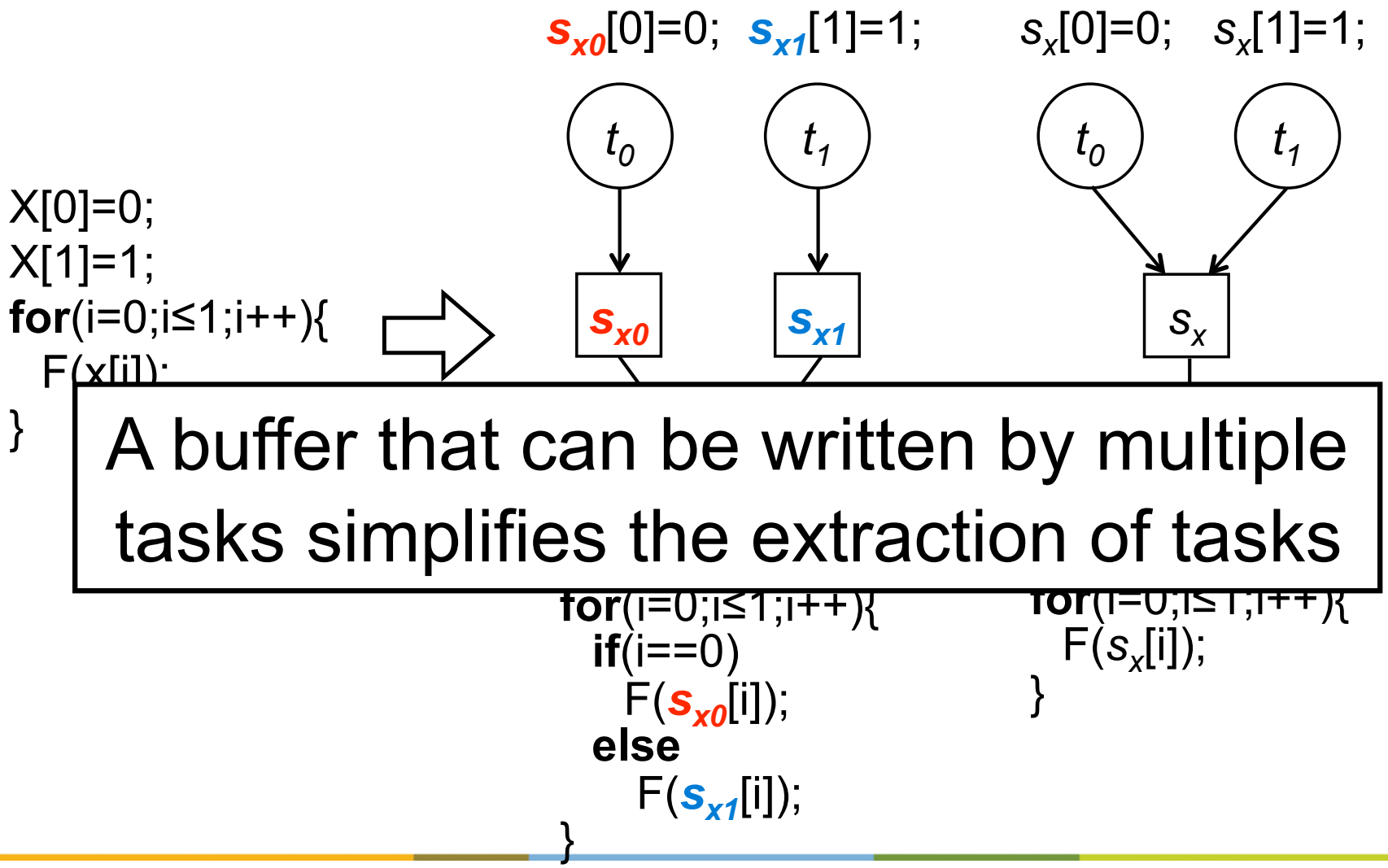

```
for(i=0;i≤1;i++){
  if(i==0)
    F(sx0[i]);
  else
    F(sx1[i]);
}
```




```
for(i=0;i≤1;i++){
  F(sx[i]);
}
```

I. Extraction of tasks

- We create a task out of each assignment-statement



I. Extraction of tasks

- Application restrictions
  - Constant bounds in for-loops
  - Side-effect-free functions
    - No implicit dependencies
  - Single assignment code
  - The variables in the condition of an if-statement are iterators
  - An index-expressions is a function of its iterators
  - Arrays are accessed using an index-expression
    - **Not** limited to affine expressions

```
1.  int x[10];
2.  for (i0=0; i0 <5; i0++){
3.    x[2i0] = F0(~) + F1(~);
4.    x[2i0+1] = F2(~);
5.  }
6.  for (i0=0; i0 <5; i0++){
7.    for (i1=0; i1 <2; i1++){
8.      printf(" %i ", x[2i0-i1+1]);
9.      printf(" %i ", x[F3(i0,i1)]);
10.   }
11. }


12. int F3(int i0, int i1){
13.   if (i0==2 && i1==0)
14.     return 3;
15.   else
16.     return 2i0+i1;
17. }
```

I. Extraction of tasks

- Application restrictions
  - Constant bounds in for-loops
  - Side-effect-free functions
    - No implicit dependencies
  - Single assignment code
  - The variables in the condition of an if-statement are iterators
  - An index-expressions is a function of its iterators

```
1.  int x[10];
2.  for (i0=0; i0 <5; i0++){
3.    x[2i0] = F0(~) + F1(~);
4.    x[2i0+1] = F2(~);
5.  }
6.  for (i0=0; i0 <5; i0++){
7.    for (i1=0; i1 <2; i1++){
8.      printf(" %i ", x[2i0-i1+1]);
9.      printf(" %i ", x[F3(i0,i1)]);
10.   }
13.   if (i0==2 && i1==0)
14.     return 3;
15.   else
16.     return 2i0+i1;
17. }
```

These are sufficient restrictions to capture the behavior in a dataflow model

II. Inter-task communication via buffers

- Replace array accesses by communication via a buffer
  - Index-expression indicate the location to be accessed

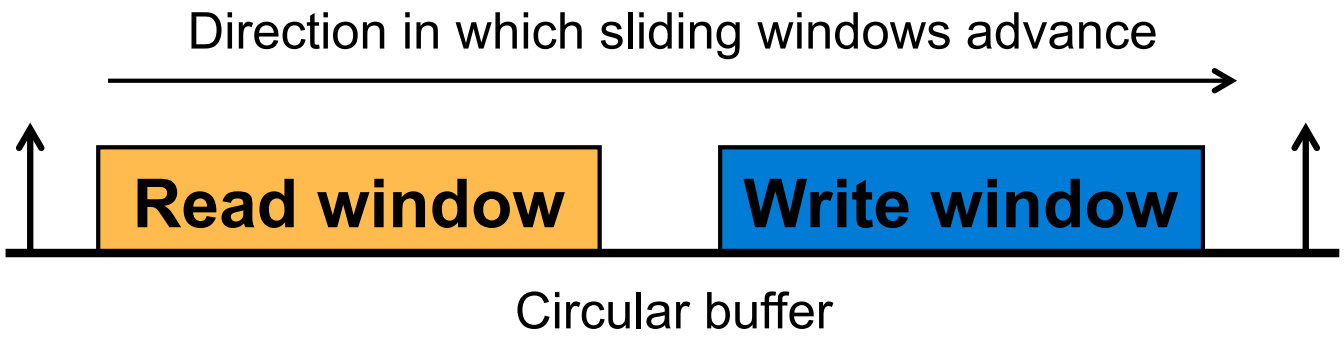
- First-in-first-out (FIFO) buffer
  - Read and write pattern must match
    - Otherwise reordering task required
    - Existing solutions require affine index-expressions [Turjan 2004 CASES]

```
1.  int x[10];
2.  for (i0=0; i0 <10; i0++){
3.    x[i0] = ~;
4.  }
5.  for (i0=0; i0 <10; i0--){
6.    printf(" %i ", x[i0]);
7.  }
```

- Alternative
  - Read from and write at locations in the buffer

II. Inter-task communication via buffers

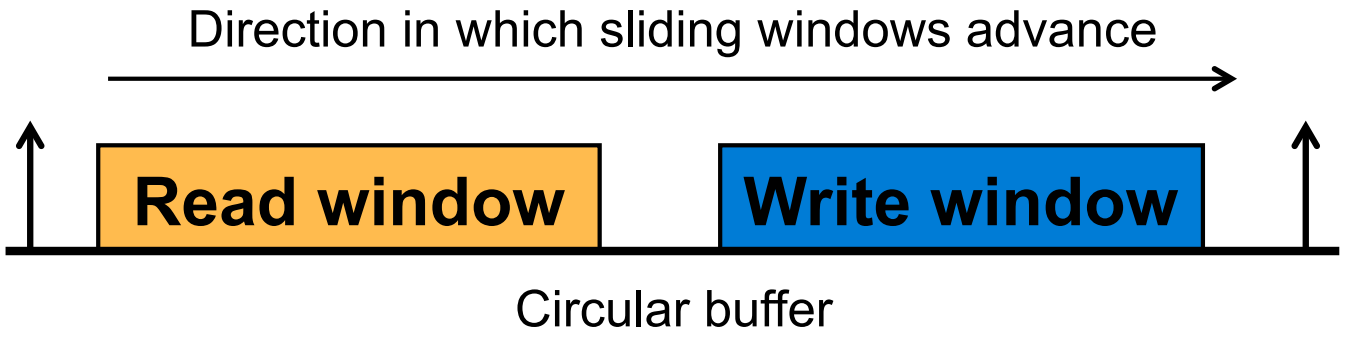
- Circular buffer
  - **Write window** and **read window**
    - Locations can be accessed in an arbitrary order
    - Do not overlap



- Each access in a window
  - Adds a location to head
  - Removes a location from the tail
- No atomic read-modify-write operations required

II. Inter-task communication via buffers

- Circular buffer
  - **Write window** and **read window**
    - Locations can be accessed in an arbitrary order
    - Do not overlap



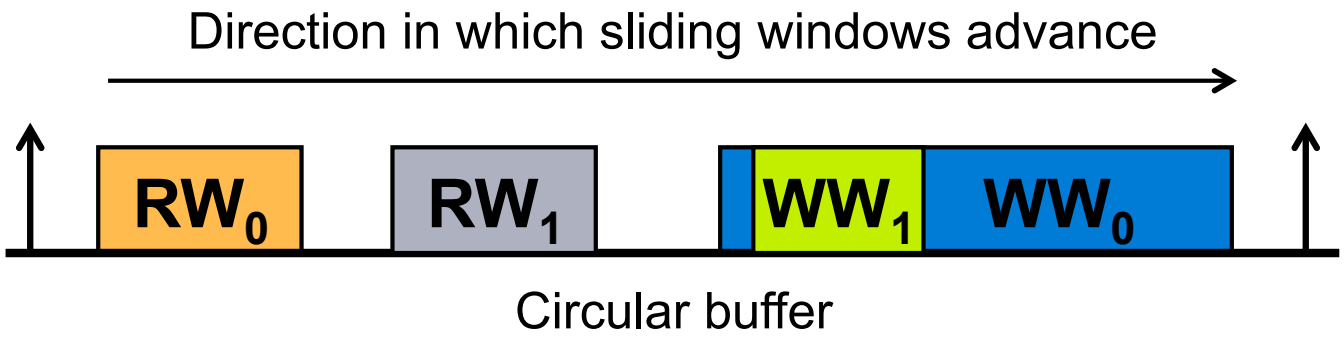
- Each
  - :
  - •

A window hides a possibly irregular access pattern

- No atomic read-modify-write operations required

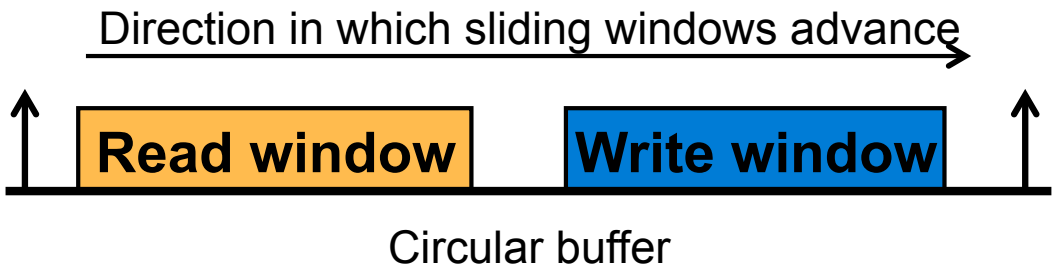
II. Inter-task communication via buffers

- Generalization with multiple write and read windows
  - –Overlap read windows (RWs)
  - –Overlap write windows (WWs)
    - Possible due to single assignment code



III. Computation of the window sizes

- An access in a window
  - Preceded by adding a location to the head
  - Succeed by removing a location from the tail
  - Window contains location to be accessed



- Initially add a number of locations to the window
  - Called the *lead-in*

- Delay removal of locations from the window
  - Called the *lead-out*

- Window size is determined by the lead-in and lead-out

IV. Insertion of communication and synchronization statements

- Replace array communication
  - Insert **read** and **write** calls that indicate the buffer to be accessed
- Add synchronization statements
  - **acquire** statement adds a location to a window
  - **release** statement removes a location from a window

  Three phases:
  - **Initial phase**
    - Acquires lead-in locations
  - **Processing phase**
    - Encapsulate communication statements
  - **Final phase**
    - Release lead-out locations

```
1.   while(1){
2.     acquire(4,x);
3.     for(i0=0;i0<5;i0++){
4.       acquire(1,x);
5.       write(sx,2i0,F1(~)+F2(~));
6.       release(1,x);
7.     }
8.     acquire(1,x);
9.     release(1,x);
10.    release(4,x);
11.  }
```

IV. Insertion of communication and synchronization statements

- Replace array communication
  - Insert **read** and **write** calls that indicate the buffer to be accessed
- Add synchronization statements
  - **acquire** statement adds a location to a window
  - **release** statement removes a location from a window

Three phases:

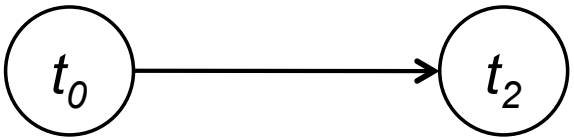
- **Initial phase**
  - Acquires lead-in locations
- **Processing phase**
  - Encapsulate communication statements
- **Final phase**
  - Release lead-out locations

```
1.   while(1){
2.     acquire(4,x);
3.     for(i0=0;i0<5;i0++){
4.       acquire(1,x);
5.       write(sx,2i0,F1(~)+F2(~));
6.       release(1,x);
7.     }
8.     acquire(1,x);
9.     release(1,x);
10.    release(4,x);
11.  }
```

Inserted synchronization is correct by construction

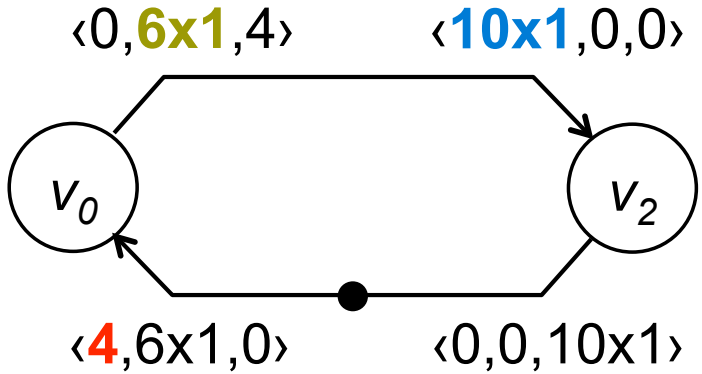
V. Computation of sufficient buffer capacities

- Regular synchronization pattern of the windows can be captured in cyclo static dataflow model
  - A token models a synchronization event, instead of a communicated value
    - Acquire is modeled by consumption from edge
    - Release is modeled by production on edge



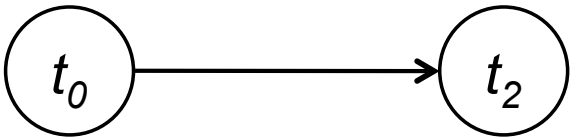

```
while(1){
  acquire(4,x);
  for(i0=0;i0<5;i0++){
    acquire(1,x);
    write(sx,2i0,F1(~)+F2(~));
    release(1,x);
  }
  acquire(1,x);
  release(1,x);
  release(4,x);
}
```

```
while(1){
  int l2=0;
  acquire(1,x);
  for(i0=0;i0<5;i0++){
    for(i1=0;i1<2;i1++){
      if(l2<9) acquire(1,x);
      printf(" %i ", read(x,2i0-i0+1))
      if(l2>0) release(1,x);
      l2++;
    }
    release(1,x);
  }
```



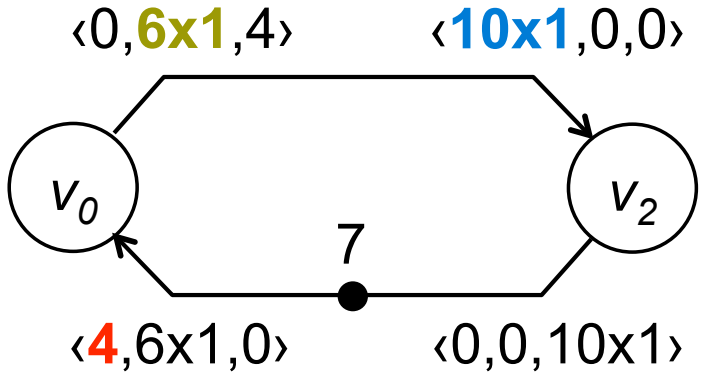
V. Computation of sufficient buffer capacities

- Regular synchronization pattern of the windows can be captured in cyclo static dataflow model
  - A token models a synchronization event, instead of a communicated value
    - Acquire is modeled by consumption from edge
    - Release is modeled by production on edge
  - Computation of sufficient buffer capacities for deadlock free execution
    - Algorithm of [Wiggers 2007 DAC] has a polynomial complexity




```
while(1){
  acquire(4,x);
  for(i0=0;i0<5;i0++){
    acquire(1,x);
    write(sx,2i0,F1(~)+F2(~));
    release(1,x);
  }
  acquire(1,x);
  release(1,x);
  release(4,x);
}
```

```
while(1){
  int l2=0;
  acquire(1,x);
  for(i0=0;i0<5;i0++){
    for(i1=0;i1<2;i1++){
      if(l2<9) acquire(1,x);
      printf(" %i ", read(x,2i0-i0+1))
      if(l2>0) release(1,x);
      l2++;
    }
    release(1,x);
  }
```



Conclusions

- Parallelization has been added to our mapping flow, in order to:
  - Meet the throughput constraint of the application
  - Derive a correct analysis model

- Automatic parallelization
  - We can extract parallelism from an application with non-affine index-expressions
  - A buffer for multiple reading and writing tasks simplifies the derivation of parallelism
  - A CSDF model is used to computed the capacities of the buffers that are large enough to guarantee deadlock free execution of the application
    - Future work: compute scheduler settings, adjust synchronization granularity

If interested, please ask for a demo of the mapping flow

Questions?